Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 8670151, 14 pages
https://doi.org/10.1155/2020/8670151

Research Article

A 3D Surface Reconstruction Method for Large-Scale Point Cloud Data

Baoyun Guo, Jiawen Wang, Xiaobin Jiang, Cailin Li, Benya Su, Zhiting Cui, Yankun Sun, and ChangLei Yang

School of Civil and Architectural Engineering, Shandong University of Technology, Zibo 255049, China

Correspondence should be addressed to Jiawen Wang; 1650947689@qq.com

Received 10 November 2019; Revised 29 June 2020; Accepted 9 July 2020; Published 6 August 2020

Academic Editor: Purushothaman Damodaran

Copyright © 2020 Baoyun Guo et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Due to the memory limitation and lack of computing power of consumer level computers, there is a need for suitable methods to achieve 3D surface reconstruction of large-scale point cloud data. A method based on the idea of divide and conquer approaches is proposed. Firstly, the kd-tree index was created for the point cloud data. Then, the Delaunay triangulation algorithm of multicore parallel computing was used to construct the point cloud data in the leaf nodes. Finally, the complete 3D mesh model was realized by constrained Delaunay tetrahedralization based on piecewise linear system and graph cut. The proposed method performed surface reconstruction on the point cloud in the multicore parallel computing architecture, in which memory release and reallocation were implemented to reduce the memory occupation and improve the running efficiency while ensuring the quality of the triangular mesh. The proposed algorithm was compared with two classical surface reconstruction algorithms using multigroup point cloud data, and the applicability experiment of the algorithm was carried out; the results verify the effectiveness and practicability of the proposed approach.

1. Introduction

The 3D surface reconstruction of point cloud data is a critical problem in the field of computer vision, and many scholars have done extensive research on it. Traditional surface reconstruction approaches of point cloud include explicit and implicit mesh reconstruction methods. Explicit methods seek to directly triangulate point cloud data, while implicit methods obtain the final triangulation by reconstructing the implicit function $f(p) = 0$ and extracting the iso-surface of the function, where p represents the triangulated point cloud data. Two categories of surface reconstruction methods have their own advantages and disadvantages [1]. The reconstruction based on explicit method is fast and has rich information of mesh, but the algorithm is limited by the amount of data. When the amount of data is large, it will cause a normal computer to run out of memory and fail to process [2–5]. The implicit method can process the point cloud data with noise but the surface details of the original model may be lost in the process of mesh reconstruction, and the algorithm usually requires massive calculation [6, 7]. Therefore, it is necessary to design an efficient and feasible mesh reconstruction scheme for large-scale point cloud data by combining the advantages of explicit and implicit methods.

1.1. Previous Work on Delaunay Triangulation and Large Point Sets. Since Delaunay triangulation is the dual of the Voronoi diagram, it can be constructed quickly in 2D or higher dimensions, so it has important application in 3D geometric reconstruction. Several classic Delaunay triangulation algorithms were tested by Su and Drysdale [8], and the performance of different algorithms was compared. Although Delaunay triangulation has been studied for many years, the mesh reconstruction problems of large-scale point sets are not trivial, including efficiency of mesh generation, the mesh reconstruction of unevenly distributed point clouds, and mesh fusion. In order to improve the efficiency of mesh generation, parallelization of Delaunay

triangulation has been proposed by many researchers. Chen et al. [9] implemented Delaunay triangulation of two-dimensional point clouds by using parallel computing. Kohout and Kolingerová [10] proposed a parallel algorithm for Delaunay triangulation based on a randomized incremental insertion algorithm parallelized with swaps. The algorithm has a simple implementation and good efficiency. Batista et al. [11] conducted multi-core parallel computation on several geometric algorithms and achieved good results. Although the Delaunay triangulation algorithm based on parallel computing can improve the efficiency of mesh reconstruction to a certain extent, the algorithm occupies a large amount of computer memory for large-scale point cloud datasets [12].

For the 3D surface reconstruction of large-scale point cloud data, Isenburg et al. [13] proposed a Delaunay triangulation based on streaming computation, which realized the mesh reconstruction of large-scale point cloud data. However, the algorithm could not achieve the surface reconstruction of point clouds with uneven distribution. Lo [14] used the parallel incremental insertion of partitions to construct the Delaunay tetrahedron, which improved the efficiency of surface reconstruction, but the algorithm still occupied a large amount of computer memory when dealing with greater than 50 million points [12]. Wu et al. [15] dynamically scheduled mesh reconstruction of point clouds and mesh merge threads in the form of streaming computation, which reduced the memory usage of the computer. However, the algorithm is not suitable for the situation of unevenly distributed point clouds and it is difficult to merge the mesh between independent blocks. Vu [16] used graph cut algorithm to realize the mesh reconstruction of multi-view stereo point clouds based on the idea of divide and conquer, but this algorithm has the problem of insufficient performance when processing very large-scale point cloud data [17]. Li [18] created a quadtree index to divide the point cloud data and assigned the mesh reconstruction of segmented point clouds and mesh merging to different processors to reduce memory, improving the efficiency of mesh reconstruction through multi-core parallel computing. However, the algorithm is not suitable for point clouds with special conditions such as cluster, uneven, and band distribution. Cao et al. [19] proposed a 3D Delaunay triangulation with GPU parallel acceleration, which can greatly reduce the run time of surface reconstruction, but it did not provide an effective solution in memory management. Boltcheva and Lévy [20] realized the surface reconstruction based on the parallel constrained Voronoi cells and extracted the manifold mesh. This algorithm has good time performance and can realize the mesh reconstruction of 100-million-point cloud in a few minutes, but they note in their paper that it cannot reconstruct mesh of 100-million-point cloud on 16 GB computer.

1.2. Previous Work on Constrained Delaunay Tetrahedralization. The concept of constrained Delaunay triangulation in two-dimensional space was first proposed by Lee and Lin [21]. They pointed out that the triangulation that obeys constraints always exists in two-dimensional space, which does not need to add additional Steiner points. And its essence is the triangulation of a set of points that comply with the constraint of edges or faces. However, not all polyhedrons can be tetrahedralized in three-dimensional constrained Delaunay tetrahedralization (such as Schonhardt's polyhedron [22]), and additional Steiner points need to be added to achieve the tetrahedralization.

Three-dimensional constrained Delaunay tetrahedralization was first proposed by Shewchuk, who gave sufficient conditions for the existence of constrained Delaunay tetrahedralization and proposed that the Steiner points only need to be added to the input edges [23]. Subsequently, some related algorithms of constrained Delaunay tetrahedralization with Steiner points inserted at the boundary were proposed [24, 25]. Si [26] pointed out that there are two key problems in the three-dimensional Delaunay tetrahedralization. One problem is the determination of Steiner points, and the other problem is the Delaunay refinement algorithm to avoid very short edges. A series of work was carried out in response to these problems and achieved good results. In terms of determining Steiner points, Shewchuk [23] inserted the Steiner point into the midpoint of the boundary edge, but this method would cause a lot of extra Steiner points to exist. Si [25, 27] adaptively inserted Steiner points on the boundary edge according to a series of new point criteria. This method greatly reduces the number of Steiner points. After all boundary edges have been restored, some related algorithms for restoring missing boundary facets without inserting Steiner points were proposed, which have good implementability [24, 25]. In terms of Delaunay refinement, Shewchuk [28] extended two-dimensional Delaunay refinement algorithm [29] to three dimensions and realized the mesh refinement in three-dimensional space. Subsequently, Si [26] proposed a constrained Delaunay mesh refinement algorithm based on the work of Shewchuk, which set a restriction of the local mesh sizing information for the insertion of the circumscribed center of the tetrahedron. Shewchuk and Si [30] dealt with the problem of generating too many small tetrahedrons in the small angle area of boundary by the Delaunay tetrahedral mesh generator using the constrained Delaunay tetrahedralization. Among them, a TetGen mesh generator [31] developed by Si was used to generate tetrahedral meshes and perform Delaunay tetrahedralization, which has been widely used in 3D reconstruction. It should be pointed out here that the implementation of constrained Delaunay tetrahedralization of this paper is based on TetGen. TetGen is written by C++, which can be compiled into a library and applied to the 3D reconstruction program under Windows operating system in this paper.

1.3. Previous Work on Graph Cuts. Graph cut algorithm is a very useful and popular energy optimization algorithm, which has the advantages of fast speed and strong applicability and has been widely used in the field of computer vision [32, 33]. In the field of mesh reconstruction, many researchers have realized the surface reconstruction of point cloud data based on the graph cuts.

Hornung and Kobbelt [34] and Lempitsky and Boykov [35] used the graph cut algorithm to implement mesh reconstruction of point cloud data based on the minimum surface frame proposed by Boykov and Funkalea [36]. Labatut [37, 38] used the three-dimensional Delaunay triangulation to obtain the Delaunay tetrahedron of the 3D point set and used the graph cut energy optimization function to realize the mesh extraction of Delaunay tetrahedron. Similarly, some related surface reconstruction algorithms applied graph cuts to extract triangles of Delaunay tetrahedron [39–42]. Although these algorithms differ in the construction of the function term of the graph cut energy equation, they all used the global graph cut optimization to realize the mesh reconstruction of point cloud. Similar to the extraction of surface from Delaunay tetrahedron using graph cut algorithm, after implementing the constrained Delaunay tetrahedralization of overlapping area point cloud, this paper uses graph cut energy optimization algorithm to label the tetrahedron and extract the final surface.

With the increase of the problem size of large-scale point sets from thousands of points to millions of points, it is necessary to propose an efficient and feasible scheme to achieve surface reconstruction. Considering the advantage that the kd-tree index can achieve the same number of point clouds in each area after being divided without being affected by the point cloud distribution [43, 44], based on the idea of divide and conquer and the advantages of the traditional Delaunay triangulation algorithm, this paper presents a Delaunay triangulation algorithm based on kd-tree index for surface reconstruction of large-scale point cloud data. Through kd-tree index construction of point cloud, 3D Delaunay triangulation algorithm is used to reconstruct the mesh of divided point cloud data. Based on constrained Delaunay tetrahedralization and graph cut algorithm, the overlapping regions of each block are merged to achieve 3D surface reconstruction of large-scale point cloud data.

2. Background

2.1. Piecewise Linear System. The piecewise linear system [31] (abbreviated as PLS) is a finite collection χ consisting of points, lines, faces, and polyhedral (collectively called cells). Its dimension represents the maximum dimension of the polyhedron, written as $\dim(\chi)$, and underlying space of χ is defined as $|\chi| = \cup_{P\in\chi} P$, where $\cup_{P\in\chi} P$ is the area to be triangulated. The PLS needs to meet the following attributes:

(1) If cells $P \in \chi$, all faces of P are contained in χ

(2) If cells $P, Q \in \chi$, then the intersection between them belongs to the cells combination in χ

(3) If $\dim(P \cap Q) = \dim(P)$, $P \neq Q$, then $P \subseteq Q$, $\dim(P) < \dim(Q)$

The PLS definition does not allow its monomers to illegally intersect. For example, the two segments of χ can only intersect at a common vertex, which belongs to χ; the intersection of two faces of χ can only be a shared segment or vertex or the set of the vertices and line segments, and it lies within χ. In order to use the boundary of PLS as the constraint of Delaunay tetrahedralization, PLS allows monomers (points, lines, faces) to float in the polyhedral region, so that boundary conditions can be used in these monomers. An example of a PLS is shown in Figure 1.

A k-simplex σ is a convex hull of $k+1$ affine independent points. For example, 0-simplex, 1-simplex, 2-simplex, and 3-simplex correspond to vertices, boundary lines, triangles, and tetrahedrons, respectively. A simplex S is a finite set of simplex, and the underlying space is the union set of all simplexes in S.

2.2. Constrained Delaunay Tetrahedralization. Assuming that χ is the finite set of PLS in $\mathbb{R}^3$, the constrained Delaunay tetrahedralization of χ is divided into the simplicial complex T that obeys edges and faces of χ, and each simple in T needs to satisfy one of the following constraint Delaunay criteria:

(1) The circumscribing sphere of the tetrahedron σ does not contain any vertices of χ, or these vertices are not visible relative to the inside of the tetrahedron. As shown in Figure 2(a), there are the face $F \subseteq \chi$, the vertices $a, b, c, p \in F$, the edge $ab \in \chi$. d, q and c, p are not visible due to the segmentation of face F and edge ab, the circumscribed sphere of tetrahedral $abcd$ contains point q, but the vertex q is not visible from inside tetrahedron $abcd$, and the circumscribed sphere of tetrahedral $abcd$ does not contain the vertices p, which satisfies the constrained Delaunay criterion.

(2) Suppose s is a triangle in T, if s belongs to the only tetrahedron in T, or s is the common plane of the tetrahedron t_1 and t_2, and the circumscribed sphere of t_1 (t_2) does not contain the vertices of t_2 (t_1), then, s satisfies the constraint Delaunay criterion, as shown in Figure 2(b).

Since not every PLS can be segmented by tetrahedron, additional Steiner points can be added so that the Delaunay tetrahedralization of PLS is always present [23, 26]. Si [26] constructed a new PLS χ' consistent with the underlying space of χ by inserting Steiner points at the boundary of PLS χ, so that all edges of χ' are Delaunay edges, satisfying the strongly Delaunay condition proposed by shewechunk [23]. In this paper, the constrained Delaunay tetrahedralization is used to realize the reconstruction of the tetrahedron of the overlapping area in the mesh merging. The specific process of constrained Delaunay tetrahedralization is as follows.

First, the segments and facets in PLS are scattered into small segments or triangles, and the Delaunay tetrahedralization of the discrete vertex set and constraint point set is performed; Second, the existence of each discretized small segment or small triangle in the tetrahedral mesh is checked. And if it does not exist, it is restored by using the constrained Delaunay tetrahedral criterion. Then, if the segments and triangles of the boundary are encroached by other nodes, they are subdivided according to the corresponding new point criterion until all the small triangles and segments are not encroached by other nodes. Finally, the quality of the mesh cells is optimized according to the given quality limit parameters (e.g., radius-edge ratio between the radius of the

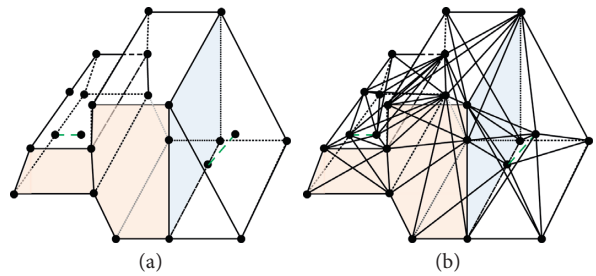


Figure 1: Schematic diagram of piecewise linear system. (a) PLS sample, (b) Delaunay triangulation of PLS.

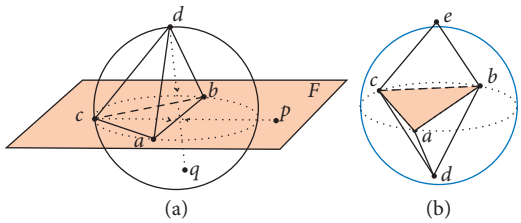


Figure 2: Constraint Delaunay criterion.

circumscribed sphere of tetrahedron and the length of its shortest edge and dihedral angle between two faces), and the mesh cells outside the boundary are deleted.

2.3. Surface Extraction by Graph Cuts

2.3.1. Graph Cuts. Let a finite directed graph with a set of nodes $v = \{v_1, \ldots, v_n\}$ and edges ε with non-negative weights w_{ij} be denoted as $G = (v, \varepsilon)$. Each node in graph $G = (v, \varepsilon)$ is connected to two special vertices, the source s and the sink t, and the corresponding non-negative weights are s_i and t_i. The graph cut problem is to find two sets of disjoint point sets $\mathcal{S}$ and $\mathcal{T}$ such that $\mathcal{S} \cup \mathcal{T} = v$, $\mathcal{S} \cap \mathcal{T} = \varnothing$.

The graph cut cost of s-t is the sum of the capacities of all edges and has the following expression:

$$\text{cost}(\mathcal{S}, \mathcal{T}) = \sum_{\substack{v_i \in \mathcal{S} \setminus \{s\} \\ v_j \in \mathcal{T} \setminus \{t\}}} w_{ij} + \sum_{v_i \in \mathcal{S} \setminus \{s\}} t_i + \sum_{v_j \in \mathcal{T} \setminus \{t\}} s_i, \quad (1)$$

where $\text{cost}(\mathcal{S}, \mathcal{T})$ is a similarity measure between point sets $\mathcal{S}$ and $\mathcal{T}$.

The smallest s-t cut problem is to find the set of points $\mathcal{S}$ and $\mathcal{T}$ with the smallest $\text{cost}(\mathcal{S}, \mathcal{T})$, and it is equivalent to the problem of calculating the maximum flow from s-t [32, 36, 45]. Such a graph cut algorithm can be regarded as the binary labeling problem of nodes.

2.3.2. Surface Extraction. After obtaining the results of constrained Delaunay tetrahedralization, graph cut algorithm is used to mark Delaunay tetrahedron as inside and outside and then extract triangles between adjacent tetrahedrons with different labels to obtain the final mesh model. Let the nodes correspond to Delaunay tetrahedrons, where each node is connected to source s and sink t, and the edges correspond to the common triangle between adjacent tetrahedra. Thus, a directed graph cuts $G = (v, \varepsilon)$ can be constructed. According to the visibility order of each node in the point sets, the label assignment problem can be transformed in to an energy minimization problem. The energy function is constructed as follows:

$$E(S) = E_{\text{data}}(S) + E_{\text{quality}}(S), \quad (2)$$

where S is the extracted surface of model, $E_{\text{data}}(S)$ represents the data item and is used to measure visibility information of each node, and $E_{\text{quality}}(S)$ represents the surface quality item and is used to the smooth the sharp triangles on the mesh surface.

The mesh merge algorithm in this paper is based on the method of Vu [16], which is shown in Section 3.3. During the surface extraction process, Vu performed constrained Delaunay tetrahedralization on overlapping area of the mesh and extracted the surface by graph cuts to realize mesh merging. Let $E_{\text{data}}(S) = \sum_{v_i \in S \setminus \{s\}} t_i + \sum_{v_j \in T \setminus \{t\}} s_j$, $E_{\text{quality}}(S) = \sum_{v_i \in S \setminus \{s\}, v_j \in T \setminus \{t\}} w_{ij}$; the sets S and T, respectively, represent tetrahedrons in front of or behind the common triangle. They calculated the constraint weight based on constrained triangular facets and the normal vector of triangular mesh vertex and calculated the quality weight by using the quality term $w_{ij} = W_{\text{qual}} \times (1 - \min\{\cos\varphi, \cos\psi\})$ proposed by Labatut et al. [37] ($W_{\text{qual}} > 0$, φ and ψ are angles of the common triangles of adjacent tetrahedra and circumscribed spheres of tetrahedron), so as to obtain the relevant mesh S_1 with the constrained triangle located on the surface. Then, they removed the extra Steiner points to obtain the surface S_2 and used the common edge of overlapping and nonoverlapping areas to cut surface S_2 to obtain the triangular mesh that could eventually be seamlessly joined with the mesh of nonoverlapping areas.

3. Mesh Reconstruction Pipeline

The basic principle of divide and conquer idea is to decompose a problem into smaller subproblems, which are independent of each other and have the same properties as the original problem. By finding the solution of the subproblems, the solution to the original problem can be obtained. This paper proposes a surface reconstruction algorithm for processing large-scale point cloud data based on the idea of divide and conquer. The basic idea is to use the spatial data structure to reduce the number of point clouds processed by the computer at one time and reduce the memory usage of the computer. The overall flow of the surface reconstruction algorithm for large-scale point cloud based on kd-tree is shown in Figure 3.

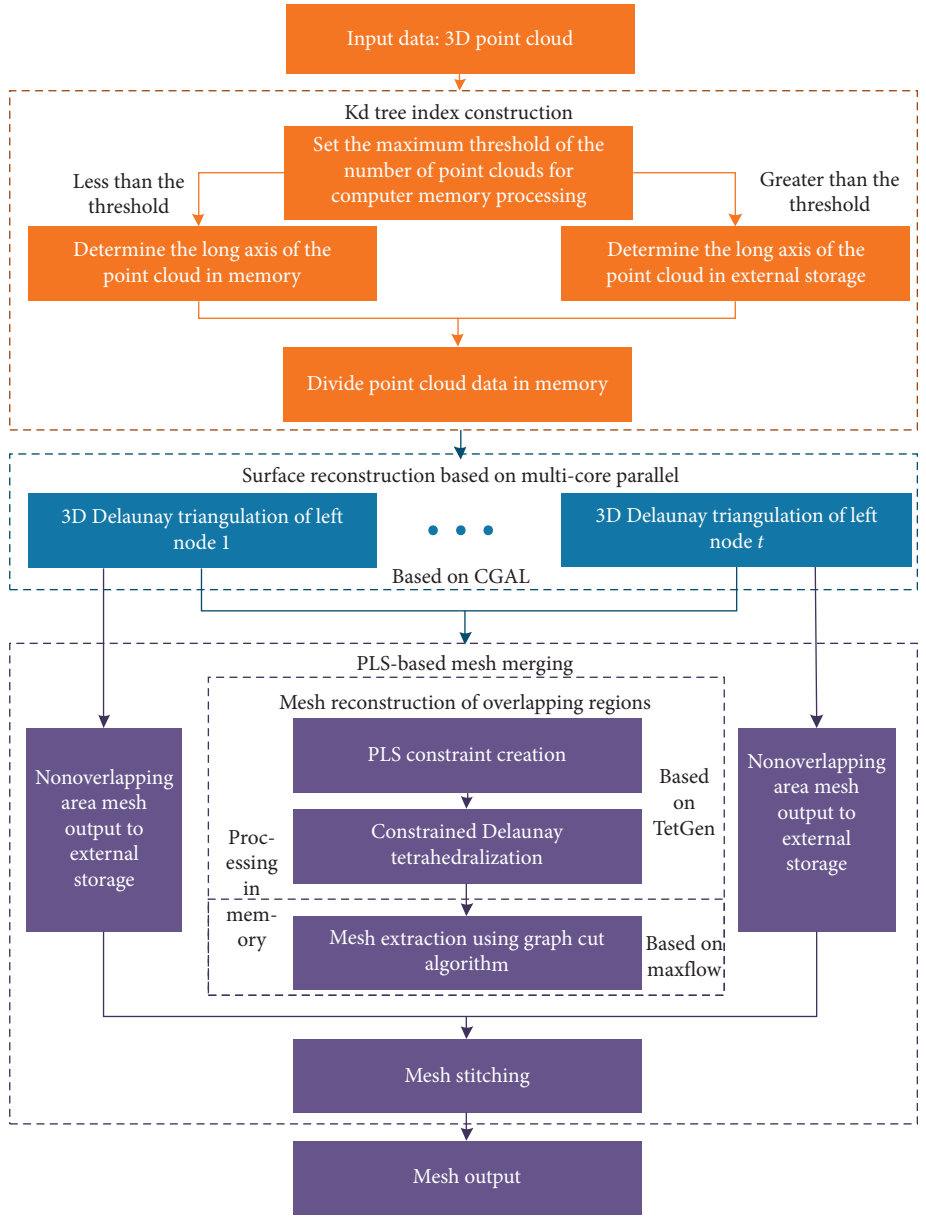


Figure 3: Algorithm flow chart.

3.1. Kd-Tree Index Construction. Delaunay triangulation of non-uniformly distributed point cloud data takes longer CPU time than uniformly distributed point cloud data, and Delaunay triangulation of non-uniformly distributed point cloud data may cause the existence of long thin tetrahedrons [44]. These long thin tetrahedrons need to be deleted and reconstructed, which wastes a lot of time. Therefore, in order to improve the efficiency of mesh reconstruction, it is necessary to divide large-scale point cloud data into uniform and evenly distributed subpoint clouds.

The large-scale point cloud data is divided according to the traditional grid rules. It is difficult to ensure the uniform distribution of point cloud data between grids after partition (as shown in Figure 4(a)), which challenges the mesh reconstruction of point cloud and mesh merging. To solve this problem, kd-tree is used to divide the point cloud data uniformly and iteratively based on the long axis of the bounding rectangle of the partitioned point cloud. The bounding rectangle of divided point cloud data is recalculated before each division, so that the original point cloud is divided into multi-block point cloud data with consistent number and uniform distribution. The division result is shown in Figure 4(b). The basic process of creating kd-tree index is to take 3D point cloud data as the root node, divide point cloud data into two equally distributed groups according to the long axis of the bounding rectangle of point cloud data in each division, and divide iteratively until the point cloud number of a single leaf node is less than a threshold. The specific process of kd-tree index construction is shown in Figure 5. Assume that the points number of point cloud dataset P is N, and the upper threshold of the minimum point cloud subset P_i is set to m.The threshold m is manually set, which is limited by the maximum number of point clouds that the computer's memory can handle. Kd-

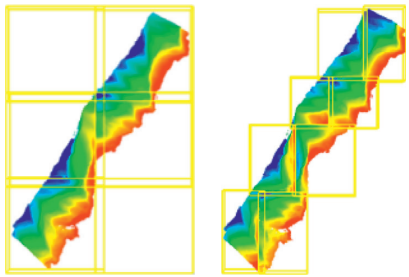


Figure 4: Schematic diagram of point cloud division. (a) Regular grid division, (b) Kd-tree division.

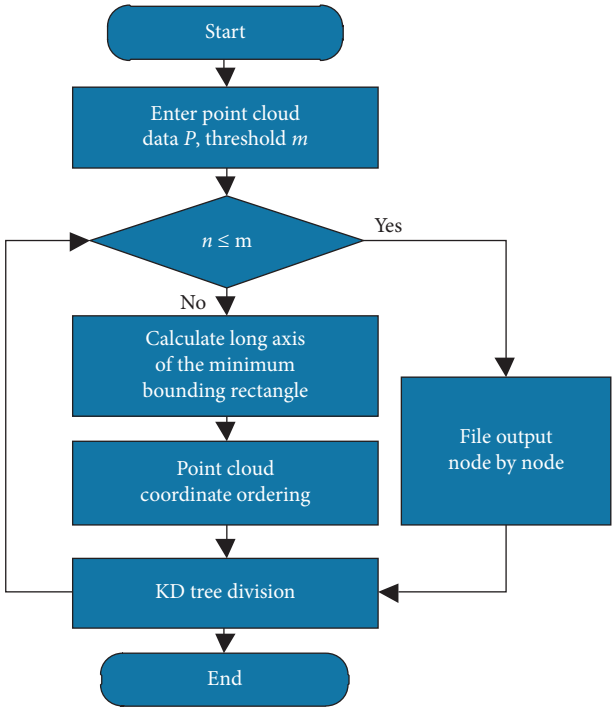


Figure 5: Kd-tree build flow chart (where n is the points number in a node, and its initial value is N; the threshold m refers to the upper threshold of the minimum point cloud subset).

tree index is adopted to evenly divide point cloud data into $t = \text{ceil}(N/m)$ point sets (ceil() is an up-round function).

An example of creating a kd-tree index of point cloud is shown in Figure 6. The specific steps of the algorithm are as follows:

Step 1: Calculate the minimum bounding rectangle of the point cloud in X-Y plane and divide the point cloud data uniformly according to the long axis of its minimum bounding rectangle. Firstly, r_1 and r_2 are computed, where $r_1 = \text{ceil}(t/2)$ and $r_2 = t - r_1$ are, respectively, the number of subsets contained in the left leaf node and the right leaf node. Then, the long axis of the point cloud bounding rectangle is determined. When x-axis of the bounding rectangle is longer than y-axis, the points are sorted from minimum to maximum by x-coordinate, and the point cloud is vertically divided on the x-axis by the i-th ($i = r_1 \times m$) point. Similarly, when the y-axis is longer than the x-axis, the point cloud is vertically divided on the y-axis.

Step 2: After the division, the point cloud data of the parent node is deleted. The number of points in the left leaf node is $i = r_1 \times m$, and the number of points in the right leaf node is $n_2 = N - n_1$. Then, it is determined whether the number of points in each leaf node after division is less than the threshold m. If the condition is true, the point cloud data in the leaf node is exported. Otherwise, repeat the above operations on the divided subset of point cloud and loop until the number of points in each leaf node is less than or equal to m.

Step 3: In order to avoid the boundary effect of local surface reconstruction, external boundary points are added to the divided point cloud. Set the number of point clouds in the overlapping area of the boundary (the number of overlapping point clouds in the experiment is $m/10$), traverse each leaf node and adjacent leaf nodes, and divide qualified points into each leaf node.

In order to reduce the usage of computer memory during the kd-tree index construction of large-scale point cloud, the maximum threshold of the number of point clouds processed by kd tree in computer memory is set. When the number of points is larger than the threshold, the long axis of the point cloud is determined in the external storage. The data is temporarily recalled during the point cloud partitioning process, and the leaf node data is stored in the corresponding file only after the index is built.

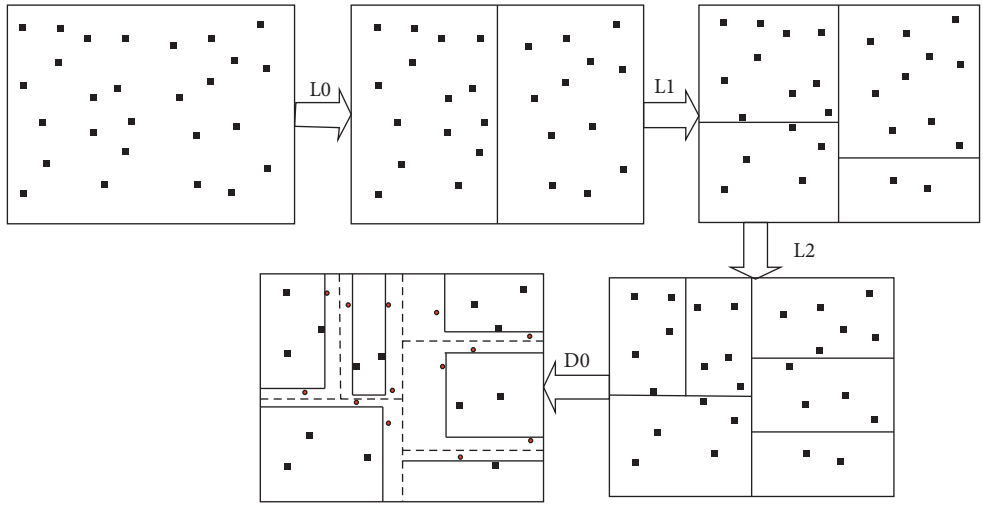
3.2. Surface Reconstruction. As the incremental insertion algorithm in 3D Delaunay triangulation is easy to implement and occupies less computer memory, this paper uses the 3D Delaunay incremental insertion algorithm based on multicore parallel computing to reconstruct mesh of point cloud in each leaf node after kd-tree index construction. The main steps of the algorithm are as follows:

Step 1: Construct an initial cuboid which contains all three-dimensional point cloud data and record the vertices of the cuboid. Connect any four vertices on the cuboid that can form a tetrahedron, and construct multiple initial Delaunay tetrahedrons in the cuboid.

Step 2: For the newly inserted three-dimensional point p, identify and find all tetrahedrons with the circumscribed ball containing p, delete these tetrahedrons, and preserve the boundaries of these tetrahedrons to generate Delaunay cavity polyhedron.

Step 3: A new tetrahedron is formed by connecting p to the vertices of the boundary of the cavity polyhedron. The validity of the new tetrahedron with p as the vertex was verified according to the 3D Delaunay Circumsphere Claim (hollow sphere criterion), the new tetrahedron was finally determined, and the adjacency relationship of the tetrahedron was updated.

Step 4: Repeat the above steps until all points are inserted, remove the tetrahedron related to the vertex of the original cuboid, and extract the triangular mesh.



L0: the first layer
L1: the second layer
L2: the third layer
D0: the addition of overlapping area
The red dot: overlapping point

Figure 6: Point cloud kd-tree division ($N = 27$, $m = 5$ as an example).

3.3. PLS-Based Mesh Merging. After conducting surface reconstruction for every point cloud set, multiple triangulated irregular network models can be obtained. Due to the redundancy of triangles in overlapping regions among these triangular mesh models, it is necessary to perform mesh reconstruction on the triangular mesh of the overlapping regions. The mesh merging algorithm of multiple blocks mesh is realized based on the method proposed by Vu [16]. In their paper, collision detection is performed on triangular mesh with overlapping regions, and the mesh vertices of the overlapping regions and the triangular ring of nonoverlapping regions adjacent to the overlapping regions are extracted as PLS [17]. The constrained Delaunay tetrahedralization [31] and graph cut algorithm [38] performed mesh reconstruction on the overlapping region and realized the mesh merging with triangulation of the nonoverlapping region. Taking Dragon's laser point cloud data as an example, the mesh merging process is shown in Figure 7.

Step 1: overlapping areas detection. Assume that a triangle ΔABC in the triangular mesh model will be merged; construct its outer bounding ball $B = (G, r)$, where the center G of the sphere is the centroid of ΔABC, and the radius r is calculated as follows:

$$r = k \times \max\{GA, GB, GC\}, \quad k > 1, \tag{3}$$

where k is the empirical weight and its value corresponds to the detection quality of overlapping areas of the triangular mesh model. In this paper, $k = 1.1$ is taken. By constructing the outer bounding spheres of triangular surfaces in mesh models of different leaf nodes and using kd-tree to accelerate the collision detection of outer bounding spheres, the intersection of outer bounding spheres can be obtained. The set of mesh triangles corresponding to the outer bounding sphere with intersection is the overlapping region of the mesh models.

Step 2: the creation of PLS constraint. After overlap detection, the triangular mesh in the overlapped region is extracted. The triangular mesh ring R_1 of nonoverlapping regions which are adjacent to overlapping mesh regions with common mesh vertices and an indirectly adjacent triangular mesh ring R_2 were extracted (R_1 and R_2 are collectively referred to as R). R_1 was used to create PLS and R_2 was used for merging mesh, as shown in Figure 7(d). In order to reduce memory usage, the mesh of the overlapping area and triangular mesh rings R are stored in the memory; the remaining mesh of nonoverlapping areas is saved in external storage, which is only used in the last stitching step. PLS were created by eliminating triangular mesh in overlapping regions, extracting mesh vertices in overlapping regions, and detecting and eliminating self-intersecting mesh triangles in triangle ring R. It includes mesh vertices in overlapping regions and triangle ring R in nonoverlapping regions.

Step 3: mesh reconstruction of overlapping regions. According to the PLS constraint created in Step 2, the constrained Delaunay tetrahedralization is constructed by using TetGen [31] to reconstruct the constrained Delaunay tetrahedron; the triangulation of the overlapping region is extracted from the constrained Delaunay tetrahedron by the graph cut algorithm.

Step 4: merging mesh. Since the triangular mesh of overlapped regions and the triangular mesh rings of adjacent non-overlapped regions have common boundaries and vertices after mesh reconstruction, the complete triangular mesh can be generated only by splicing them together with the remaining mesh of non-overlapped regions in the external storage.

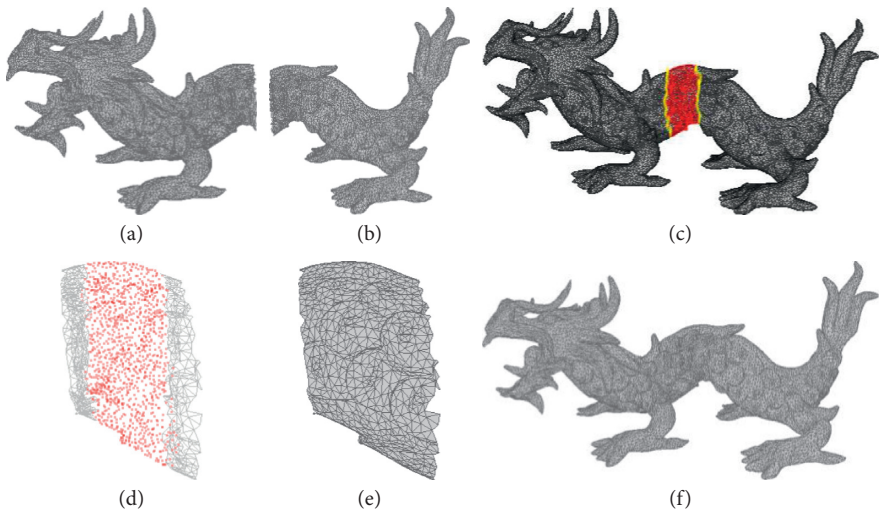


Figure 7: Schematic diagram of mesh merging. (a) Input mesh 1, (b) Input mesh 2, (c) detection of overlapping area, (d) the creation of PLS, (e) mesh reconstruction of overlapping regions, (f) mesh merging.

4. Experimental Results

In this paper, C++, the Computational Geometry Algorithms Library (CGAL) [46], TetGen [31], and maxflow [32] are used to implement a large-scale point cloud reconstruction code. In order to verify the effectiveness and applicability of the algorithm, firstly, seven point cloud datasets with different amounts are used to compare the time and memory of the algorithm in this paper with the classical Ball-Pivoting algorithm [5] and the state-of-the-art Restricted Voronoi Cells in Parallel (RVCP) algorithm proposed by Boltcheva and Lévy [20], which are both popular Delaunay triangulation algorithms in mesh reconstruction. And the 100-million-point cloud was conducted as qualitative analysis of mesh reconstruction and mesh merging. The number of point clouds in the seven datasets is 1 million, 5 million, 7 million, 10 million, 20 million, 50 million, and 100 million, respectively. Then, the algorithm in this paper is compared with Ball-Pivoting and RVCP algorithm in terms of mesh accuracy and completeness. Finally, three different types of point cloud datasets are used to evaluate the applicability of the algorithm: the dataset of Buddha statue with simple structure, the dataset of large indoor scene, and the dataset of complex ancient buildings.

4.1. The Qualitative Analysis. Seven sets of point cloud data of 1 million, 5 million, 7 million, 10 million, 20 million, 50 million, and 100 million are used to compare the mesh reconstruction time and peak memory of the proposed algorithm with Ball-Pivoting and RVCP algorithm, so as to verify the mesh construction efficiency of the proposed algorithm. And the performance of the algorithm in this paper is analyzed in detail with a 100-million-point cloud.

The time consumption and space consumption of the proposed algorithm mainly include kd-tree construction, surface reconstruction, and mesh merging. Therefore, the total time complexity of the algorithm is $O(n_1 \times t) + O(n_1) + O(n_2 \times (t-1))$, where n_1 is the number of vertices, t is the number of blocks in the point cloud, and n_2 is the number of mesh cells. And the total space complexity of the algorithm is $2O(n_1) + O(n_2)$. The time complexity and space complexity of the algorithm in this paper are both linear. Since the time complexity and space complexity of the Ball-Pivoting algorithm are also linear [5], and we cannot determine the complexity of the RVCP algorithm, the memory usage and time consumption of each algorithm are recorded to compare the execution efficiency of the algorithm. Figure 8 shows the variation trend of time and memory with the number of point clouds in the 3D surface reconstruction process of the three algorithms. Figure 8(a) shows the time consumption of each algorithm in the mesh reconstruction of seven point cloud datasets, and Figure 8(b) shows the peak computer memory when each algorithm processes different amounts of point cloud data.

As can be seen from Figure 8, the Ball-Pivoting algorithm can process millions of point cloud data quickly with lower memory, but the algorithm does not support the reconstruction of more than 20-million-point cloud data. The algorithm proposed in this paper and RVCP algorithm can handle hundreds of millions of point cloud data, and the time consumption of the algorithm proposed in this paper is basically the same as that of RVCP algorithm and Ball-Pivoting algorithm in processing millions of point cloud data. When processing more than 20-million-point cloud data, the time consumption of the algorithm in this paper is better than RVCP algorithm proposed by Boltcheva et al. As can be seen from Figure 8(b), the memory occupation of the proposed algorithm is approximately linearly related to the number of point clouds; when processing 100-million-point cloud data, the memory usage of RVCP algorithm exceeds 14 GB, while the memory usage of the proposed algorithm is less than 10 GB. The memory occupation of the algorithm in this paper is obviously better than RVCP algorithm. Figure 8 shows that the proposed algorithm can achieve efficient 3D

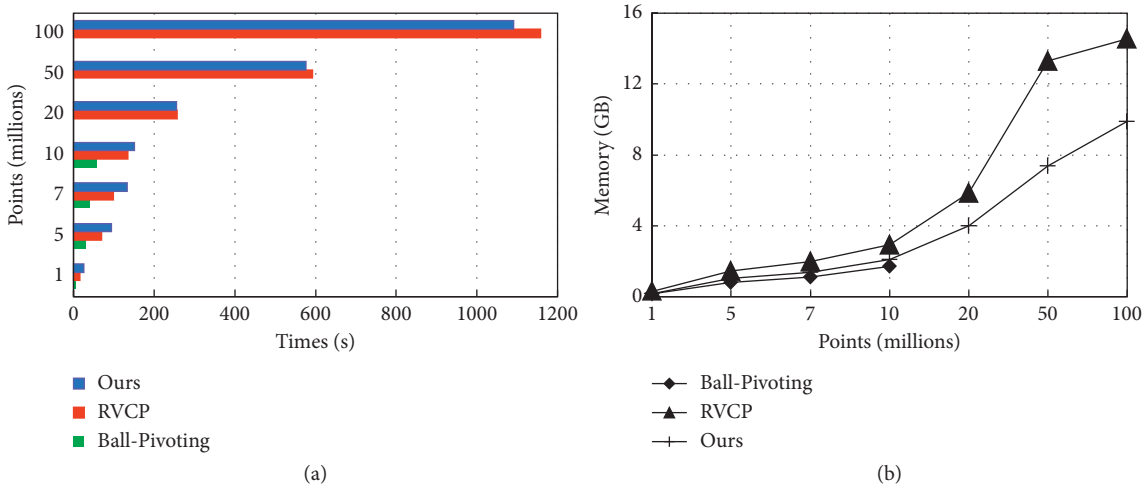


Figure 8: Comparison of experimental results of different amounts of point clouds. (a) Time consumption comparison, (b) memory usage comparison.

modeling of large-scale point clouds (100-million-level) with lower computer memory.

In order to further test the performance of the algorithm in this paper, a group of point cloud data of an urban area in HeFei was selected for testing. The point cloud data was produced from UAV tilt images, which contained 102,116,723 points in total. The algorithm in this paper is used to carry out mesh reconstruction of the point cloud data, and the threshold value $m = 200{,}000$ is manually set arbitrarily according to the amount of point cloud data that the computer can process, thus dividing the point cloud into 32 equal blocks. The result of the surface reconstruction is shown in Figure 9. Figure 9(a) shows the point cloud data indexed by kd-tree. Figure 9(b) shows the reconstruction results for point clouds in leaf nodes of kd-tree; the green line represents the boundary of meshes with overlapping area. Figure 9(c) shows the results of mesh merging. As shown in Figure 9, the algorithm in this paper can also maintain the shape characteristics of buildings in urban areas by constructing a mesh of 100-million-point cloud.

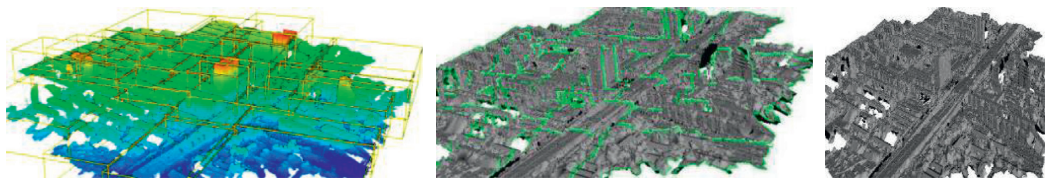
In order to intuitively demonstrate the effect of mesh merging algorithm, an overlapping area of urban data in HeFei was extracted for amplification display, as shown in Figure 10. The left graph in Figure 10 is the overlapping area of the point cloud, the middle graph in Figure 10 shows the mesh of overlapped area after surface reconstruction of two blocks divided by kd-tree, and the right graph in Figure 10 shows the mesh merging in the overlapping regions. According to the magnification and comparison of local overlapping regions shown in Figure 10, it is further verified that the mesh reconstruction pipeline can effectively eliminate mesh redundancy in overlapping regions and achieve high-quality mesh merging.

4.2. The Quantitative Analysis. In this paper, the DTU dataset [47] is used to evaluate the mesh reconstruction accuracy of the proposed algorithm, Ball-Pivoting algorithm, and RVCP algorithm. The DTU dataset is a collection of datasets for multi-view stereo evaluation and can also be used to evaluate the quality of mesh reconstruction, which consists of 124 small scenes. Each scene contains 49/69 RGB images captured by a camera in a structured light scanner and a corresponding structured light scanning point cloud.

The structured light scanning point cloud data of scene 4, scene 43, and scene 48 in the DTU dataset are, respectively, reconstructed by our method, the Ball-Pivoting, and RVCP algorithm. According to precision evaluation method of mesh reconstruction proposed in literature [47], the accuracy and completeness comparison of surface reconstruction results was conducted. The point cloud data of DTU dataset is shown in Figure 11, and the evaluated visualization results are shown in Figure 12.

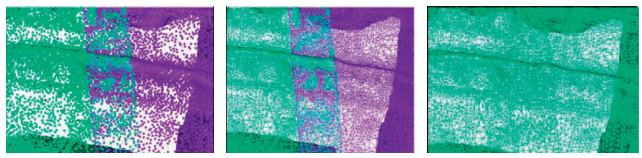
In Figure 12, white dots indicate points where the distance error from structured light scanning point cloud is 0–10 mm, red dots indicate points where the distance error is beyond the threshold, and blue and green dots indicate points that are not involved in the calculation. As can be seen from Figure 12, the red area using our algorithm is less than RVCP algorithm in terms of mesh accuracy but slightly more than Ball-Pivoting algorithm. However, Ball-Pivoting algorithm cannot process point cloud data of 20 million or more according to Section 4.1. Our algorithm is slightly better than Ball-Pivoting algorithm and RVCP algorithm in terms of mesh completeness.

In order to further demonstrate the differences of three algorithms in terms of mesh reconstruction accuracy, the mean distance errors between the mesh and the structured light scanning point cloud are, respectively, calculated, and the results are shown in Table 1. It can be seen that our algorithm and RVCP algorithm are better than the Ball-Pivoting algorithm in terms of mesh completeness, and the mesh completeness after the evaluation of surface reconstruction using our algorithm is slightly improved compared with RVCP algorithm. The mesh reconstruction accuracy of the algorithm in this paper is slightly inferior to that of Ball-Pivoting algorithm in general, but it has certain advantages compared to the RVCP algorithm for large-scale point cloud



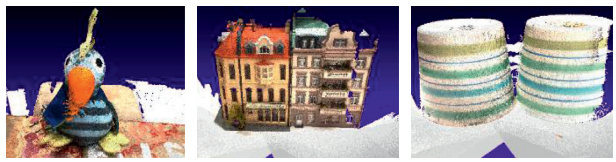
(a) (b) (c)

Figure 9: Mesh reconstruction of HeFei dataset.



(a) (b) (c)

Figure 10: Schematic diagram of mesh merging in overlapping area. (a) point cloud after division, (b) mesh reconstruction of two blocks, and (c) mesh merging.



(a) (b) (c)

Figure 11: Scan data of DTU dataset. (a) Scene 4, (b) scene 43, (c) scene 48.

data. This indicates that the proposed algorithm has better accuracy of surface reconstruction than RVCP algorithm and is more suitable than Ball-Pivoting algorithm for large-scale point cloud data.

4.3. Algorithm Applicability. In order to verify the effectiveness of the proposed algorithm in point cloud partitioning, mesh reconstruction, and mesh merging, three groups of point cloud data with different types are used to evaluate the applicability of the algorithm. They are, respectively, two point cloud datasets generated by Carl Olsson's image dataset which contains "Buddha Statue" and the large indoor scene "Eglise du dome (interior)" [48, 49] and the point cloud data for complex models generated by the image dataset "Fayu temple on mount Puto" provided by National Laboratory of Pattern Recognition in China [50].

Three groups of point cloud data are reconstructed by using our algorithm. The time consumption of each module of the algorithm and the peak memory of the whole mesh reconstruction process are counted, respectively. The visualization results of point cloud segmentation and mesh merging are shown in Figure 13, and the statistical results of time and memory are shown in Table 2. It can be seen from Table 2 that, when processing data of "Fayu temple on mount Puto" with the largest number of point clouds, our algorithm takes less than 6 minutes and consumes less than 6.5 GB of memory. As shown in Figure 13, our algorithm is used to reconstruct mesh of point cloud data such as sculptures, large indoor scenes, and complex buildings with good visualization results.

In summary, the time and space efficiency of the algorithm proposed in this paper are better than the RVCP algorithm. And compared with the Ball-Pivoting algorithm, it can better achieve mesh reconstruction of large-scale point cloud data. When processing point cloud data with 100 million datasets, the proposed method takes up less than 10 GB of memory. In terms of mesh reconstruction quality, the mesh reconstruction accuracy of the proposed method is significantly better than the RVCP algorithm. In terms of algorithm applicability, the proposed method shows good performance in large-scene indoor dataset, complex structure ancient building dataset, and simple structure Buddhist dataset. The proposed algorithm has certain advantages in memory and time consumption and accuracy of mesh reconstruction and is suitable for mesh reconstruction of point cloud data with different types, such as sculptures, large scenes, and complex buildings. Therefore, the algorithm proposed in this paper has certain validity and applicability in processing large-scale point cloud data.

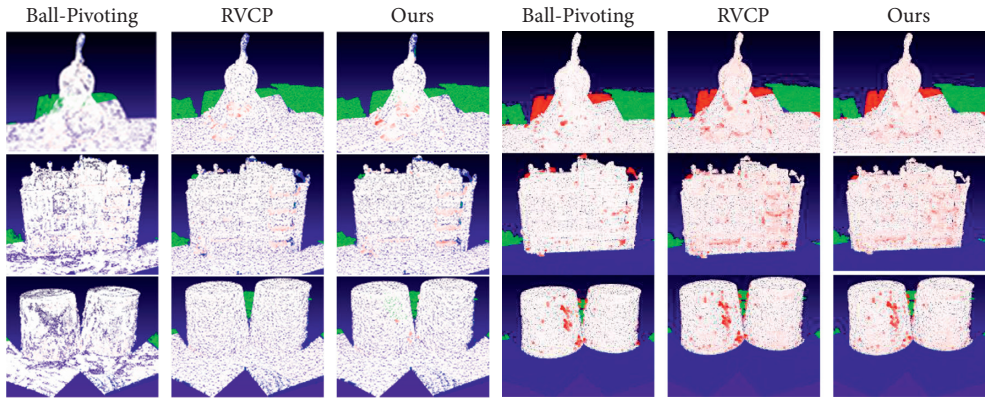


Figure 12: Accuracy and completeness evaluation results of mesh reconstruction of DTU dataset. (a) Accuracy evaluation, (b) completeness evaluation.

Table 1: Accuracy evaluation results of mesh reconstruction of DTU dataset.

	Scene	Ours	Ball-Pivoting	RVCP
Mean acc(mm)	Scene 4	29.133	8.173	30.380
	Scene 43	13.983	14.220	14.852
	Scene 48	30.784	15.238	31.097
Mean com (mm)	Scene 4	5.851	6.149	5.854
	Scene 43	4.479	4.589	4.485
	Scene 48	6.643	7.507	6.645

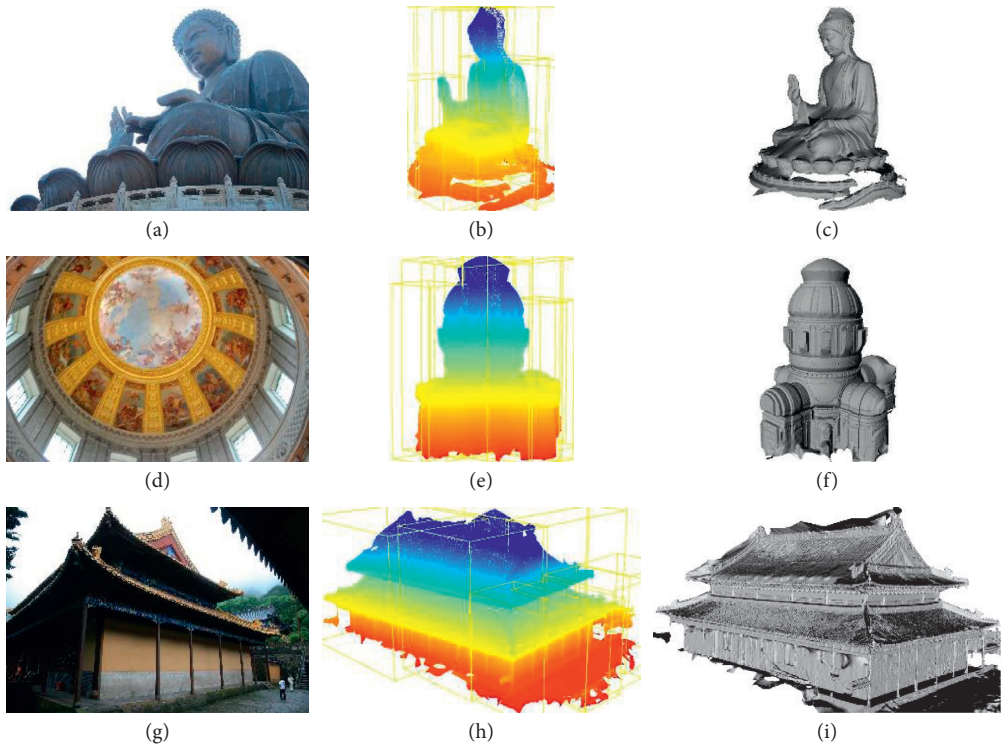


Figure 13: Visualization results of algorithm applicability experiment. (a) Image of Buddha Statue, (b) Kd-tree index construction of Buddha Statue, (c) mesh merging of Buddha Statue, (d) internal image of Eglise du dome, (e) Kd-tree index construction of Eglise du dome, (f) mesh merging of Eglise du dome, (g) image of Fayu temple, (h) Kd-tree index construction of Fayu temple, (i) mesh merging of Fayu temple.

Table 2: Experimental data statistics of algorithm applicability.

	Buddha Statue	Eglise du dome	Fayu temple
# points (in 1000)	37,115	31,769	46,195
# triangles (in 1000)	63,191	62,604	70,555
kd-tree index construction (in sec)	139.2	111.9	178.7
Surface reconstruction (in sec)	56.9	59.0	68.3
PLS-based mesh merging (in sec)	123.6	108.3	170.7
Memory (MB)	4791.8	4983.9	6298.8

5. Conclusion

Aiming to alleviate the problems of large memory occupation and long time in the process of 3D surface reconstruction of large-scale point clouds, the algorithm of mesh reconstruction has been proposed in this paper for 3D Delaunay triangulation of large-scale point clouds of more than 100 million points using a personal computer. According to the set point cloud minimum threshold m, the point clouds are adaptively divided into approximately uniform blocks which have an equal number of points; points of each block are triangulated by the multi-core parallel incremental insertion, step by step, until Delaunay triangulation of each block of points is realized. The constrained Delaunay triangulation and graph cut are used in the final mesh merging to generate the final 3D geometric model effectively. Based on the idea of divide and conquer, there is no issue in efficiently realizing the 3D reconstruction of clouds with billions of points by PCs with limited memory.

The experimental environment is Intel(R) Core(TM) i7-7700K CPU @ 4.20 GHz, eight cores with 16 GB of memory, and the operating system is Windows 10. Compared with the traditional Delaunay triangulation algorithm of Ball-Pivoting algorithm, the algorithm in this paper is not as effective in terms of the accuracy of mesh reconstruction, but it is suitable for processing point cloud data with hundreds of millions of points. Compared with the state-of-the-art methods proposed by Boltcheva and Lévy, the efficiency of the algorithm in this paper is significantly better than RVCP algorithm when dealing with 20 million or more points. Moreover, when dealing with point cloud data with 100 million points, the memory occupation of the algorithm is 4 GB less than RVCP algorithm. The memory occupation and the number of point clouds are linear in the whole process of the mesh reconstruction pipeline. Thus, the accuracy of the method proposed by this paper is better than the RVCP algorithm. The performance of the proposed method has also been tested on different types of point clouds. For urban point clouds with 100 million points, the performance of the algorithm can still be maintained and all models in this paper can maintain their detailed characteristics. The results of this study can provide some evidence that the proposed algorithm can reduce memory consumption and reconstruct 3D surface models quickly with high quality.

Observing the time consumption and memory usage in the different steps of the method, the point cloud division and mesh merging domain are the core of the algorithm. However, the outliers of point clouds need to be removed before the creation of kd-tree index due to the characteristics of incremental insertion algorithm; it also needs more time for preprocessing. Additionally, the implementation of hole fillings is not performed in this study. For future work, the goal is to realize hole-free mesh reconstruction of large-scale point cloud data more rapidly.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was funded by National Natural Science Foundation of China (Nos. 41601496 and 41701525); Key Research and Development Program of Shandong Province (No. 2018GGX106002); Shandong Provincial Natural Science Foundation (No. ZR2017LD002); and Qi Culture Research Project of Shandong University of Technology (No. 2017QWH032).

References

[1] L. D. Angelo, P. D. Stefano, and L. Giaccari, "A new mesh-growing algorithm for fast surface reconstruction," *Computer-Aided Design*, vol. 43, no. 6, pp. 639–650, 2011.

[2] T. K. Dey, J. Giesen, and J. Hudson, "Delaunay based shape reconstruction from large data," in *Proceedings of the IEEE Symposium on Parallel & Large-Data Visualization & Graphics*, pp. 257–263, San Diego, CA, USA, October 2001.

[3] N. Amenta, S. Choi, T. K. Dey, and N. Leekha, "A simple algorithm for homeomorphic surface reconstruction," *International Journal of Computational Geometry & Applications*, vol. 12, no. 1n02, pp. 125–141, 2002.

[4] N. Chrisochoides and D. Nave, "Parallel delaunay mesh generation kernel," *International Journal for Numerical Methods in Engineering*, vol. 58, no. 2, pp. 161–176, 2003.

[5] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, 1999.

[6] M. Kazhdan and H. Hoppe, "Screened Poisson surface reconstruction," *ACM Transactions on Graphics*, vol. 32, no. 3, pp. 1–13, 2013.

[7] T. K. Dey and J. Sun, "An adaptive MLS surface for reconstruction with guarantees," in *Proceedings of the Eurographics Symposium on Geometry Processing*, pp. 43–52, Vienna, Austria, 2005.

[8] P. Su and R. L. S. Drysdale, "A comparison of sequential delaunay triangulation algorithms," *Computational Geometry: Theory and Applications*, vol. 7, no. 5-6, pp. 361–385, 1997.

[9] M.-B. Chen, T.-R. Chuang, and J.-J. Wu, "Efficient parallel implementations of near delaunay triangulation with high performance fortran," *Concurrency and Computation: Practice and Experience*, vol. 16, no. 12, pp. 1143–1159, 2004.

[10] J. Kohout and I. Kolingerová, "Parallel delaunay triangulation in E3: make it simple," *Visual Computer*, vol. 19, no. 7-8, pp. 532–548, 2003.

[11] V. H. F. Batista, D. L. Millman, S. Pion, and J. Singler, "Parallel geometric algorithms for multi-core computers," *Computational Geometry*, vol. 43, no. 8, pp. 663–677, 2010.

[12] S. H. Lo, "3D delaunay triangulation of 1 billion points on a PC," *Finite Elements in Analysis and Design*, vol. 102-103, pp. 65–73, 2015.

[13] M. Isenburg, Y. Liu, J. Shewchuk, and J. Snoeyink, "Streaming computation of delaunay triangulations," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 1049–1056, 2006.

[14] S. H. Lo, "Parallel delaunay triangulation in three dimensions," *Computer Methods in Applied Mechanics and Engineering*, vol. 237–240, no. 1-2, pp. 88–106, 2012.

[15] H. Wu, X. Guan, and J. Gong, "ParaStream: a parallel streaming delaunay triangulation algorithm for LiDAR points on multicore architectures," *Computers & Geosciences*, vol. 37, no. 9, pp. 1355–1363, 2011.

[16] H. H. Vu, "Large-scale and high-quality multi-view stereo," in *Group Imagine, Laboratory of Computer Science Gaspard-Monge, Ecole des Ponts*, University of Paris-Est, Marne-la-Vallée, France, 2012.

[17] A. Kuhn and H. Mayer, "Incremental division of very large point clouds for scalable 3D surface reconstruction," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 10–18, Santiago, Chile, December 2015.

[18] J. Li, "A streaming data delaunay triangulation algorithm based on parallel computing," *Geomatics and Information Science of Wuhan University*, vol. 38, no. 7, pp. 794–798, 2013.

[19] T. Cao, A. Nanjappa, M. Gao, and T. Tan, "A GPU accelerated algorithm for 3D Delaunay triangulation," in *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pp. 47–54, San Francisco, CA, USA, March 2014.

[20] D. Boltcheva and B. Lévy, "Surface reconstruction by computing restricted voronoi cells in parallel," *Computer-Aided Design*, vol. 90, pp. 123–134, 2017.

[21] D. T. Lee and A. K. Lin, "Generalized delaunay triangulation for planar graphs," *Discrete & Computational Geometry*, vol. 1, no. 3, pp. 201–217, 1986.

[22] E. Schönhardt, "Über die zerlegung von dreieckspolyedern in tetraeder," *Mathematische Annalen*, vol. 98, no. 1, pp. 309–312, 1928.

[23] J. R. Shewchuk, "Constrained delaunay tetrahedralizations and provably good boundary recovery," in *Proceedings of the 11th International Meshing Roundtable*, pp. 193–204, Ithaca, NY, USA, 2002.

[24] J. R. Shewchuk, "Updating and constructing constrained delaunay and constrained regular triangulations by flips," in *Proceedings of the Symposium on Computational Geometry*, pp. 181–190, San Diego, CA, USA, 2003.

[25] H. Si and K. Gartner, "Meshing piecewise linear complexes by constrained delaunay tetrahedralizations," in *Proceedings of the 14th International Meshing Roundtable*, pp. 147–163, San Diego, CA, USA, 2005.

[26] H. Si, "Constrained delaunay tetrahedral mesh generation and refinement," *Finite Elements in Analysis and Design*, vol. 46, no. 1-2, pp. 33–46, 2010.

[27] H. Si and K. Gärtner, "3D boundary recovery by constrained delaunay tetrahedralization," *International Journal for Numerical Methods in Engineering*, vol. 85, no. 11, pp. 1341–1364, 2011.

[28] J. R. Shewchuk, "Tetrahedral mesh generation by delaunay refinement," in *Proceedings of the Symposium on Computational Geometry*, pp. 86–95, Paris, France, June 1998.

[29] J. Ruppert, "A delaunay refinement algorithm for quality 2-dimensional mesh generation," *Journal of Algorithms*, vol. 18, no. 3, pp. 548–585, 1995.

[30] J. R. Shewchuk and H. Si, "Higher-quality tetrahedral mesh generation for domains with small angles by constrained delaunay refinement," in *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, pp. 290–299, Kyoto, Japan, June 2014.

[31] H. Si, "TetGen, a quality tetrahedral mesh generator and three-dimensional delaunay triangulator," *Weierstrass Institute for Applied Analysis and Stochastic*, vol. 75, 2006.

[32] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124–1137, 2004.

[33] N. Vu and B. S. Manjunath, "Shape prior segmentation of multiple objects with graph cuts," in *Proceedings of the Computer Vision and Pattern Recognition*, pp. 1–8, Anchorage, AK, USA, June 2008.

[34] A. Hornung and L. Kobbelt, "Robust reconstruction of watertight 3D models from non-uniformly sampled point clouds without normal information," in *Proceedings of the Symposium on Geometry Processing*, pp. 41–50, Sardinia, Italy, 2006.

[35] V. Lempitsky and Y. Boykov, "Global optimization for shape fitting," in *Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, Minneapolis, MN, USA, June 2007.

[36] Y. Boykov and G. Funkalea, "Computing geodesics and minimal surfaces via graph cuts," in *Proceedings of the Ninth IEEE International Conference on Computer Vision*, pp. 26–33, Nice, France, October 2003.

[37] P. Labatut, J.-P. Pons, and R. Keriven, "Robust and efficient surface reconstruction from range data," *Computer Graphics Forum*, vol. 28, no. 8, pp. 2275–2290, 2009.

[38] P. Labatut, J.-P. Pons, and R. Keriven, "Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts," in *Proceedings of the 2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8, Rio de Janeiro, Brazil, October 2007.

[39] M. Jancosek and T. Pajdla, "Multi-view reconstruction preserving weakly-supported surfaces," in *Proceedings of the Computer Vision and Pattern Recognition*, pp. 3121–3128, Providence, RI, USA, June 2011.

[40] M. Jancosek and T. Pajdla, "Exploiting visibility information in surface reconstruction to preserve weakly supported surfaces," *International Scholarly Research Notices*, vol. 2014, Article ID 798595, 20 pages, 2014.

[41] D. Ma, G. Li, and L. Wang, "Rapid reconstruction of a three-dimensional mesh model based on oblique images in the internet of things," *IEEE Access*, vol. 6, pp. 61686–61699, 2018.

[42] M. Wan, Y. Wang, and D. Wang, "Variational surface reconstruction based on delaunay triangulation and graph cut," *International Journal for Numerical Methods in Engineering*, vol. 85, no. 2, pp. 206–229, 2011.

[43] C. Lemaire and J.-M. Moreau, "A probabilistic result on multi-dimensional delaunay triangulations, and its application to the 2D case," *Computational Geometry*, vol. 17, no. 1-2, pp. 69–96, 2000.

[44] S. H. Lo, "3D Delaunay triangulation of non-uniform point distributions," *Finite Elements in Analysis and Design*, vol. 90, pp. 113–130, 2014.

[45] Y. Boykov and G. Funka-Lea, "Graph cuts and efficient N-D image segmentation," *International Journal of Computer Vision*, vol. 70, no. 2, pp. 109–131, 2006.

[46] The Computational Geometry Algorithms Library, https://www.cgal.org/.

[47] H. Aanæs, R. R. Jensen, G. Vogiatzis, E. Tola, and B. Dahl, "Large-scale data for multiple-view stereopsis," *International Journal of Computer Vision*, vol. 120, no. 2, pp. 153–168, 2016.

[48] O. Enqvist, F. Kahl, and C. Olsson, "Non-sequential structure from motion," in *Proceedings of the 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 264–271, Barcelona, Spain, November 2011.

[49] C. Olsson and O. Enqvist, "Stable structure from motion for unordered image collections," in *Proceedings of the Scandinavian Conference on Image Analysis*, pp. 524–535, Ystad, Sweden, 2011.

[50] 3D Reconstruction Dataset, http://vision.ia.ac.cn/data.